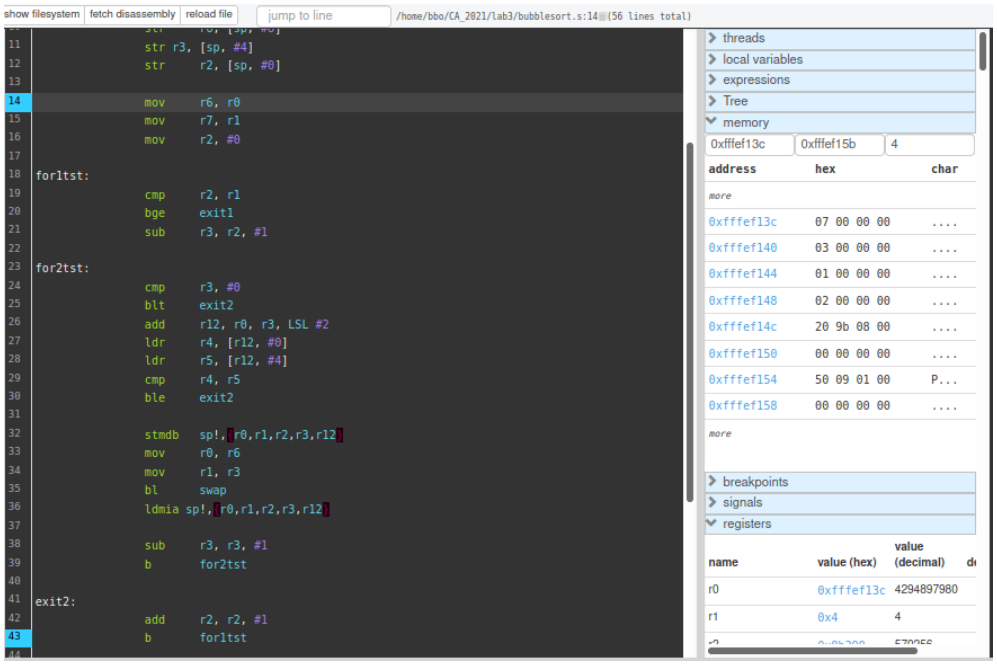
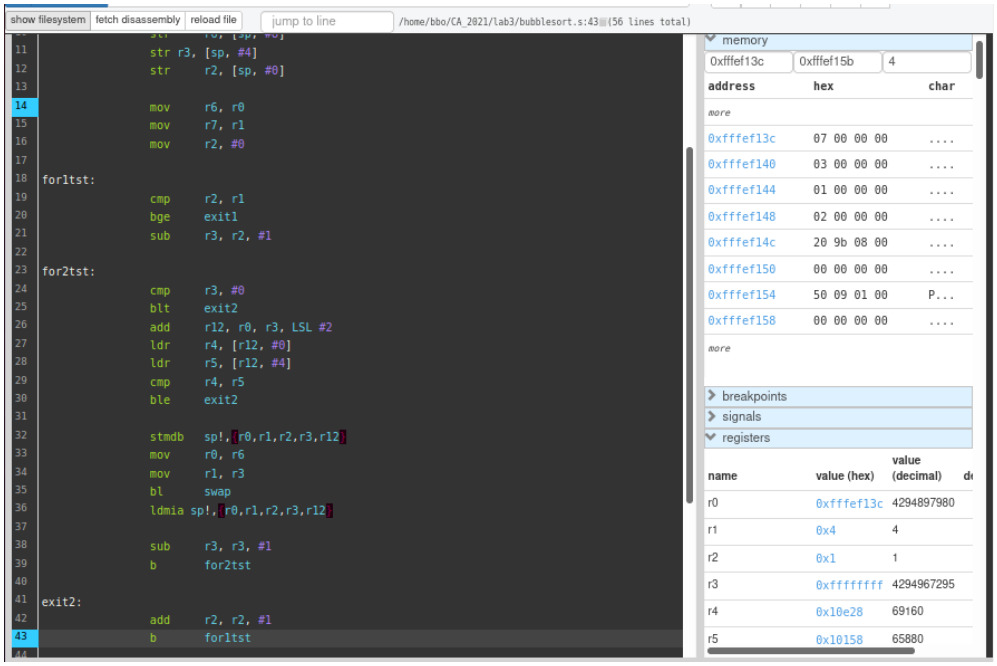
HW6 : lab2 실습

20160394 임효상

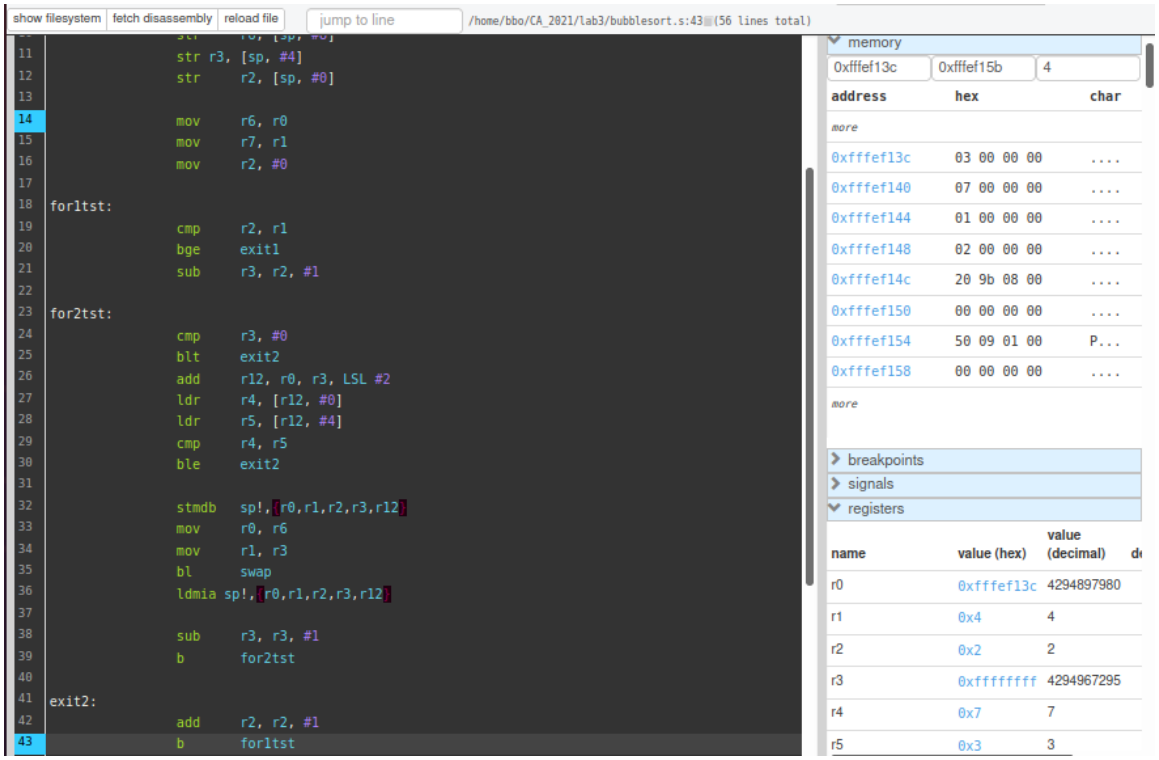
1) 실습 A



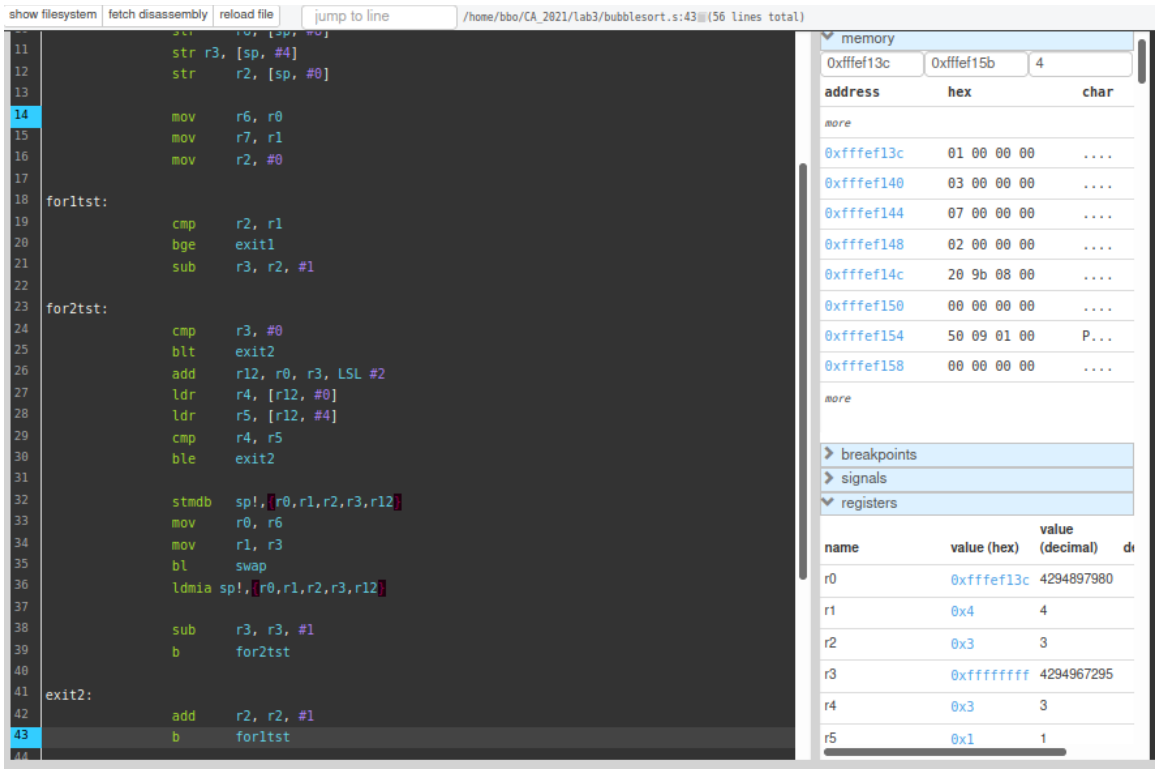
첫 outer loop 수행 전 화면이다. 배열이 7,3,1,2 순서로 저장된 상태다.



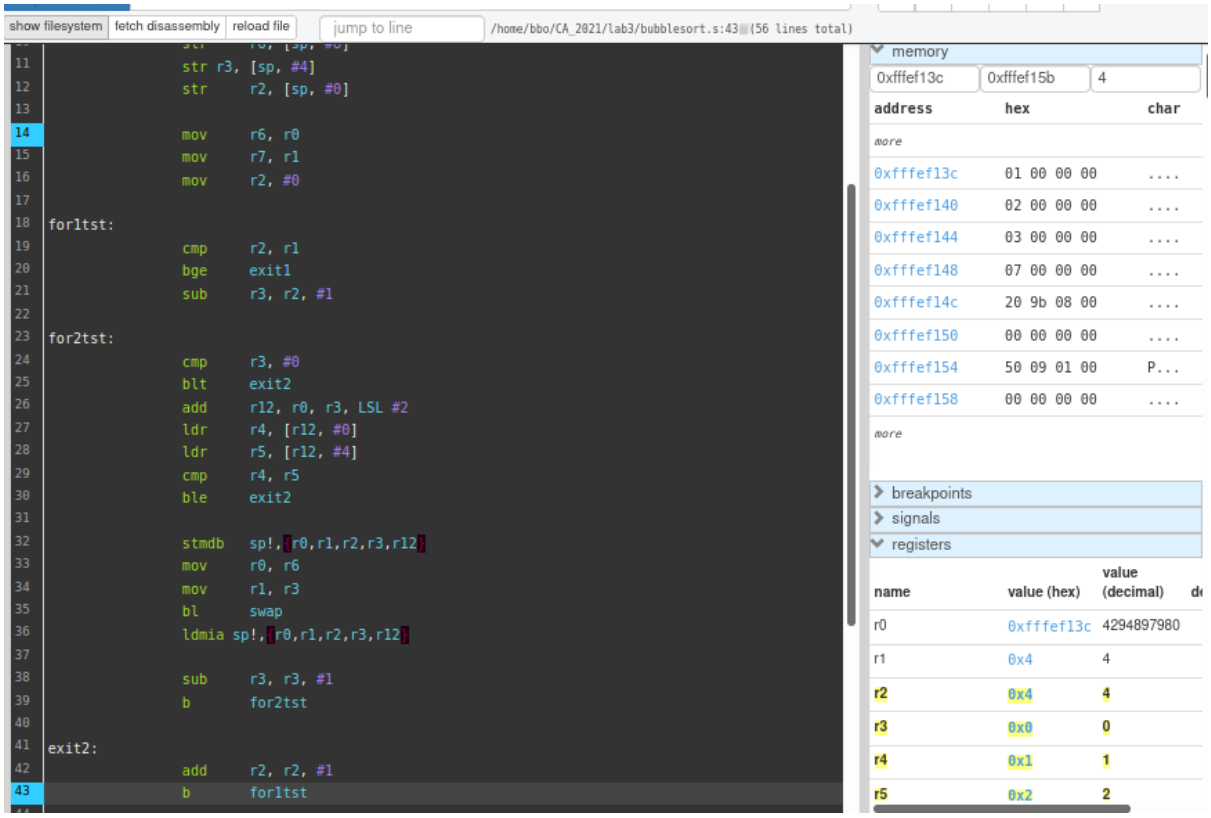
첫 번째 외부 루프(i=0) 실행 후 화면이다. j에 해당하는 r3에는 -1이, i에 해당되는 r2에는 1이 저장된다.(i+1을 했으므로) 배열은 첫번째 방이 정렬되어 7,3,1,2다.



두 번째 외부 루프(i=1) 실행 후 화면이다. j에 해당하는 r3에는 -1이, i에 해당되는 r2에는 2가 저장된다.(i+1을 했으므로) 배열은 첫번째 방과 두번째 방까지 정렬되어 3,7,1,2가 된 상태다.



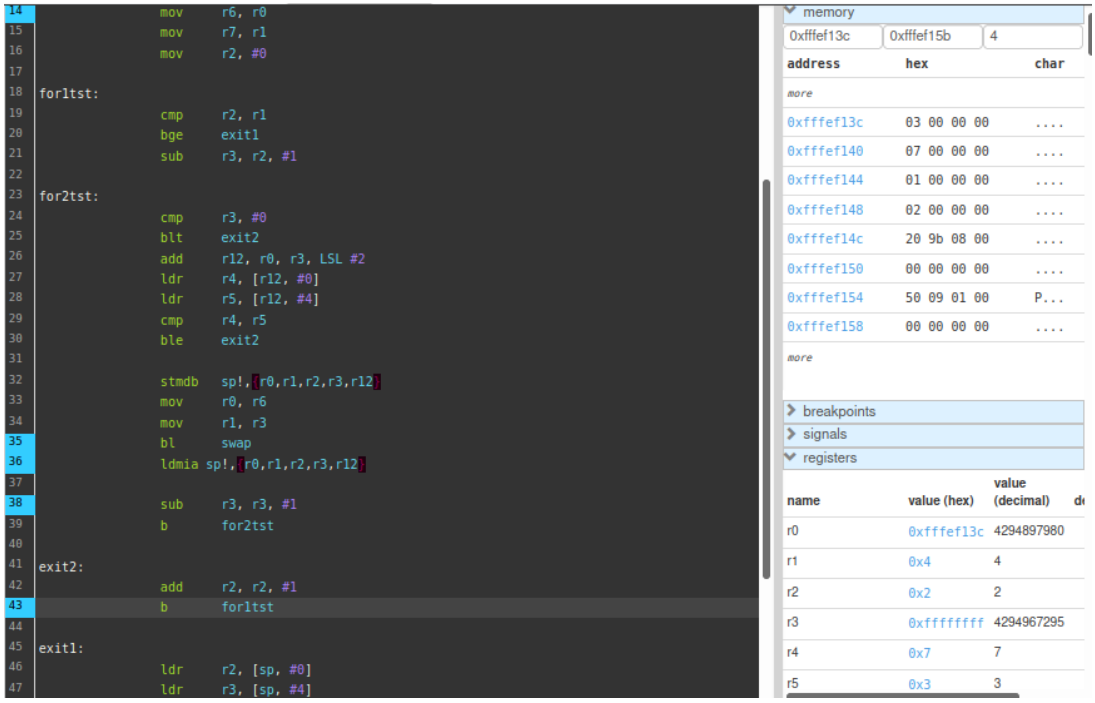
세 번째 외부 루프(i=2) 실행 후 화면이다. j에 해당하는 r3에는 -1이, i에 해당되는 r2에는 3이 저장된다.(i+1을 했으므로) 배열은 첫번째 방과 세번째 방까지 정렬되어 1,3,7,2가 된 상태다.



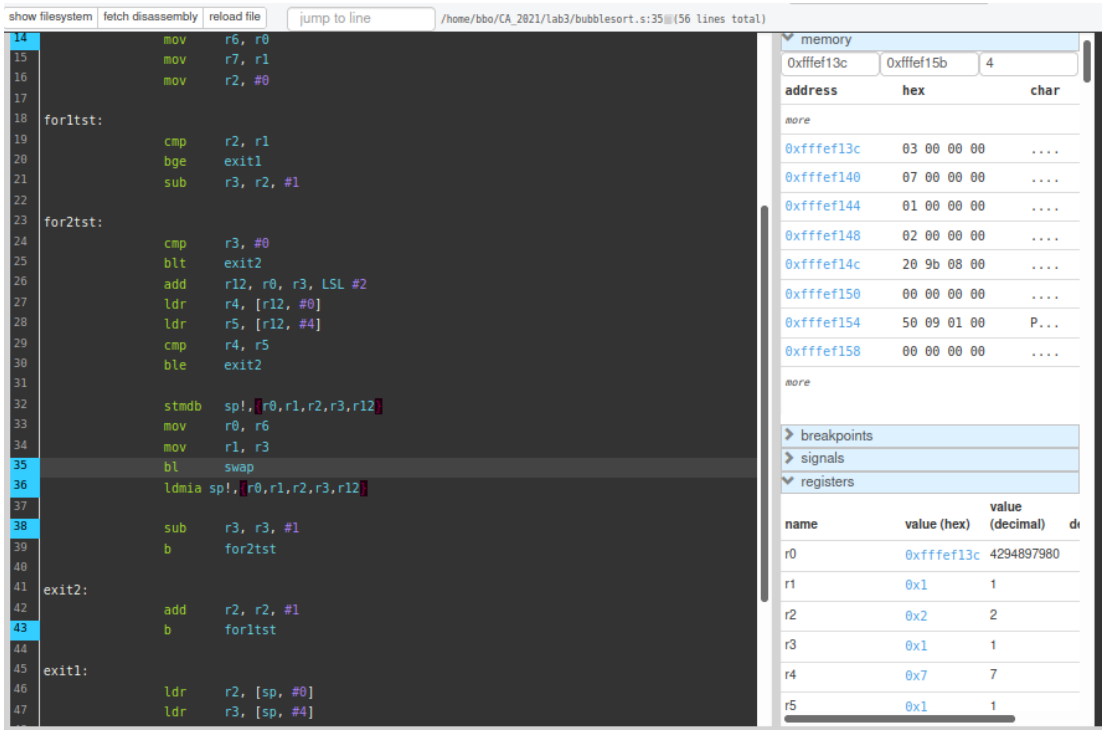
네 번째 외부 루프(i=3) 실행 후 화면이다. j에 해당하는 r3에는 0이, i에 해당되는 r2에는 4가 저장된다.(i+1을 했으므로) 배열은 모두 정렬되어 1,2,3,7가 된 상태다.

r2=r1이므로 ble 조건에 의해 루프가 끝난다.

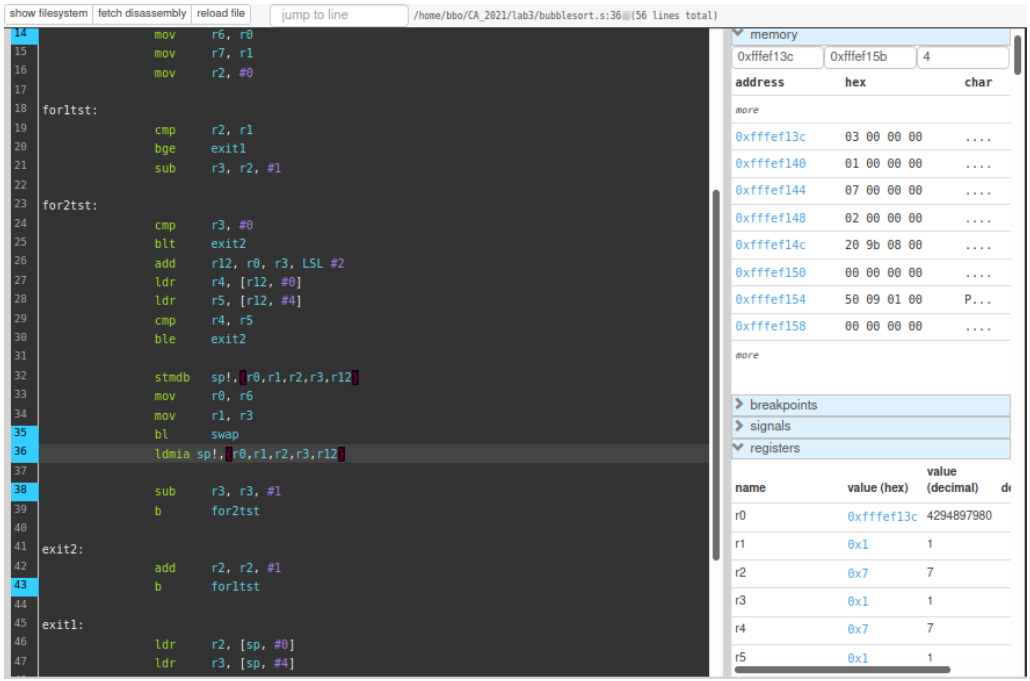
2) 실습 B



add r2,r2,#1 명령어에 의해 r2(i)가 2로 바뀐 시점이다.



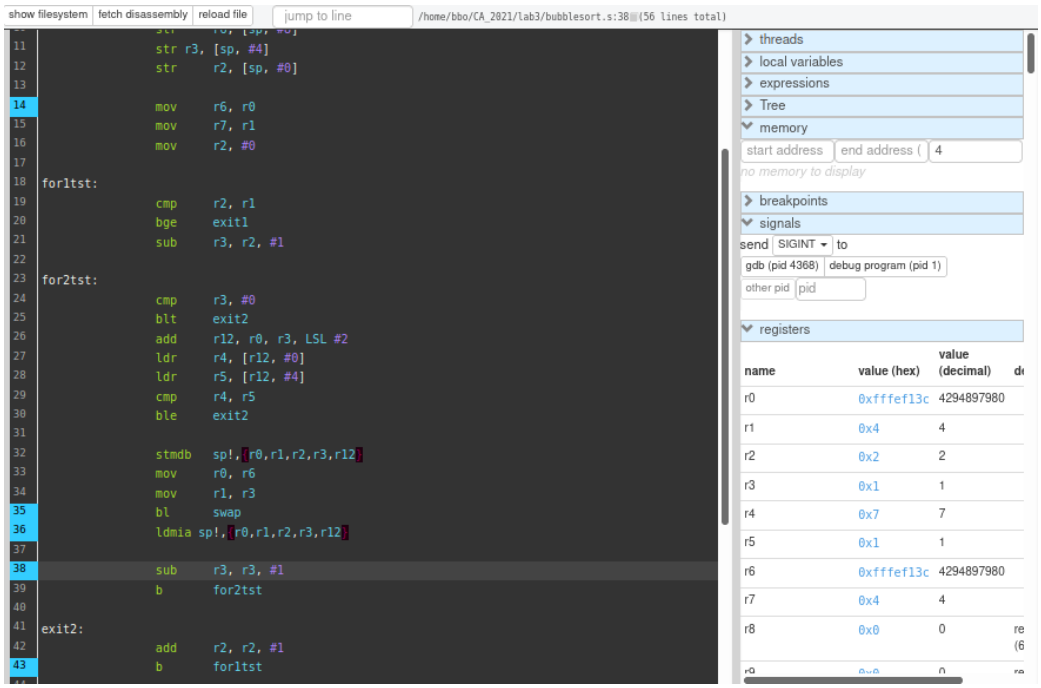
i=2인 outer loop에서 첫 번째 inner loop로 j=1인 상태이다. r4는 배열의 두번째 방(7), r5는 배열의 세번째 방(1)을 가리키고 있다. cmp이후 ble에서 branch하지 않고 swap함수를 거치게된다.



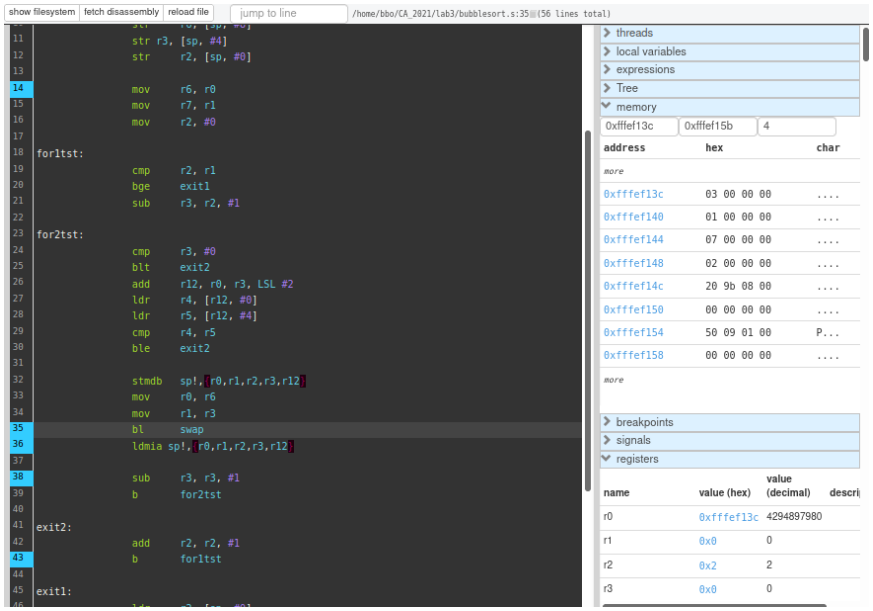
swap함수를 거친 상태다. 두번째 방이었던 7과 세번째 방이었던 1이 swap된 상태이다.

swap함수를 보면 레지스터 r2,r3를 변수로 사용하고 있다. 따라서 r2에는 두번째 방의 7, r3에는 세번째 방의 1이 저장된 것을 확인할 수 있다.

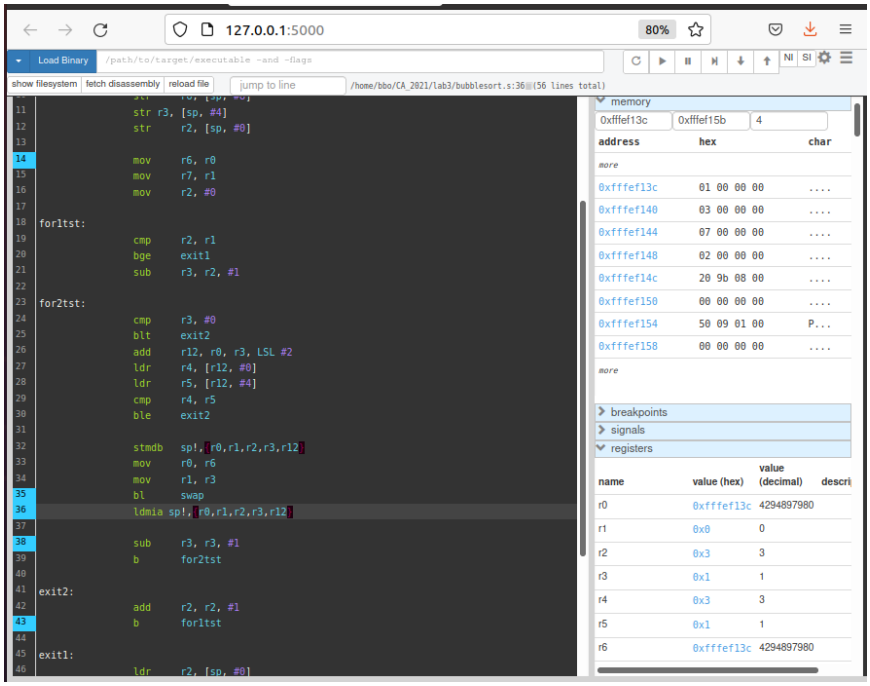
r1과 r0 역시 배열의 인덱스를 swap함수에서 사용하기위해 이전에 r0는 배열 시작주소, r1은 swap할 인덱스로 초기화 된 것을 볼 수 있다.



스택에 저장해두었던 데이터를 이용하여 swap함수로 진입하기 전 레지스터 상태를 복구했다.



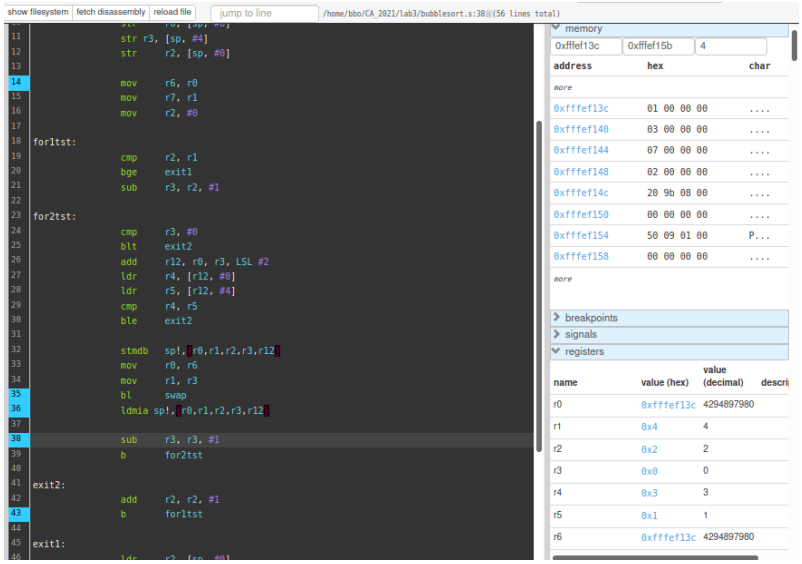
inner loop가 1회 끝나고 r3가 1 감소하여 0이 되었다. 즉 j=0이다. r4는 배열의 첫 번째 방(3), r5는 배열의 두번째 방(1)을 가리키고 있다. cmp이후 ble에서 branch하지 않고 swap함수를 거치게 된다.



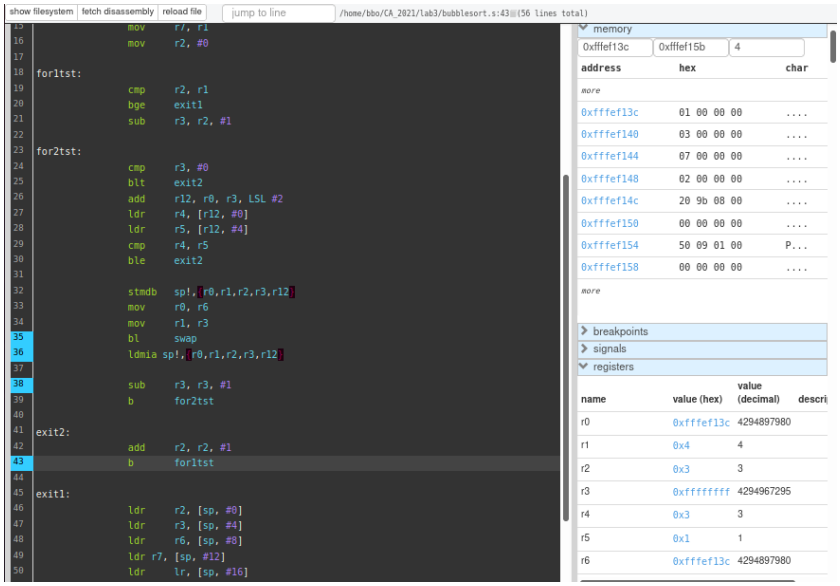
swap함수를 거친 상태다. 첫번째 방이었던 3과 두번째 방이었던 1이 swap된 상태이다.

swap함수에서 사용하기위해 r2에는 첫번째 방의 3, r3에는 두번째 방의 1이 저장된 것을 확인할 수 있다.

r1과 r0 역시 배열의 인덱스를 swap함수에서 사용하기위해 이전에 r0는 배열 시작주소, r1은 swap할 인덱스로 초기화 된 것을 볼 수 있다.

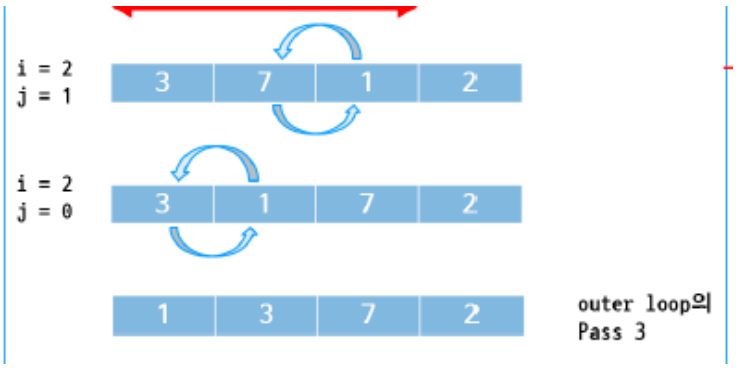


스택에 저장해두었던 데이터를 이용하여 swap함수로 진입하기 전 레지스터 상태를 복구했다.



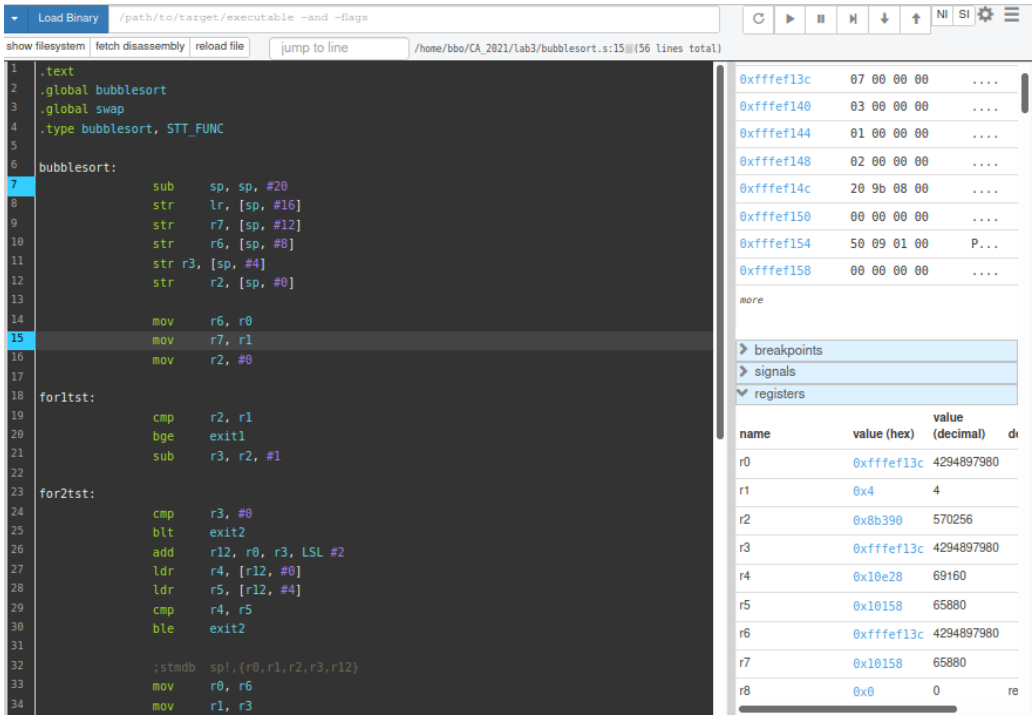
r3에서 1을 뺀 상태로, r3=-1이므로, cmp r3,#0와 blt 에서 branch된 모습이다.

inner loop가 모두 끝나고 outer loop가 1회 종료되면서 r2는 1이 더해서 3이 된다.

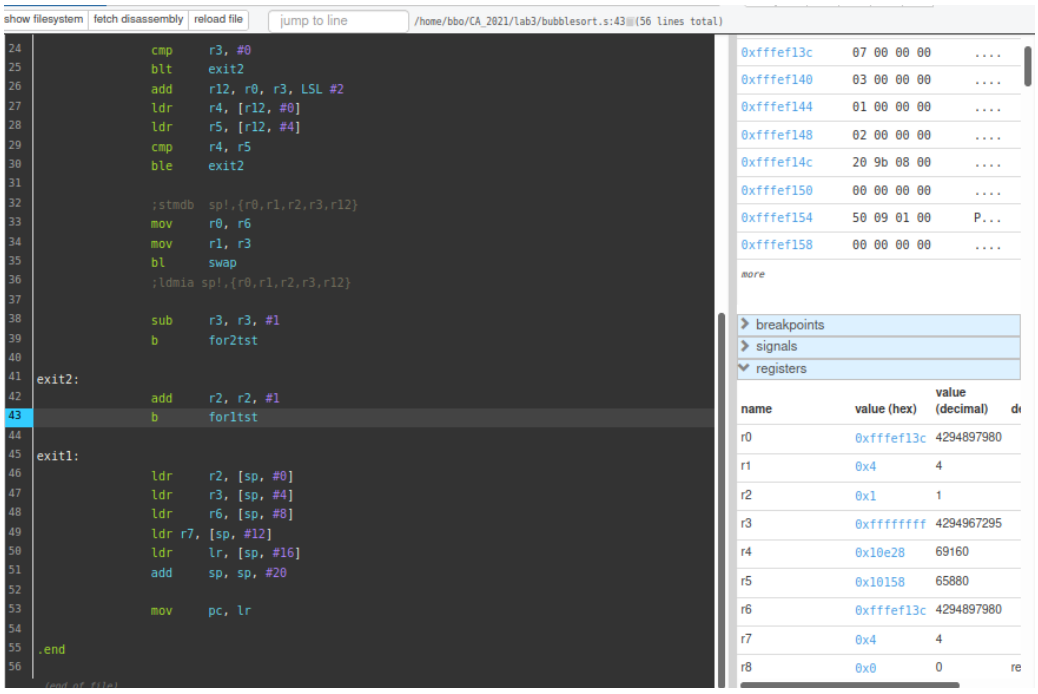


arr에 제대로 저장된 것을 확인할 수 있다.

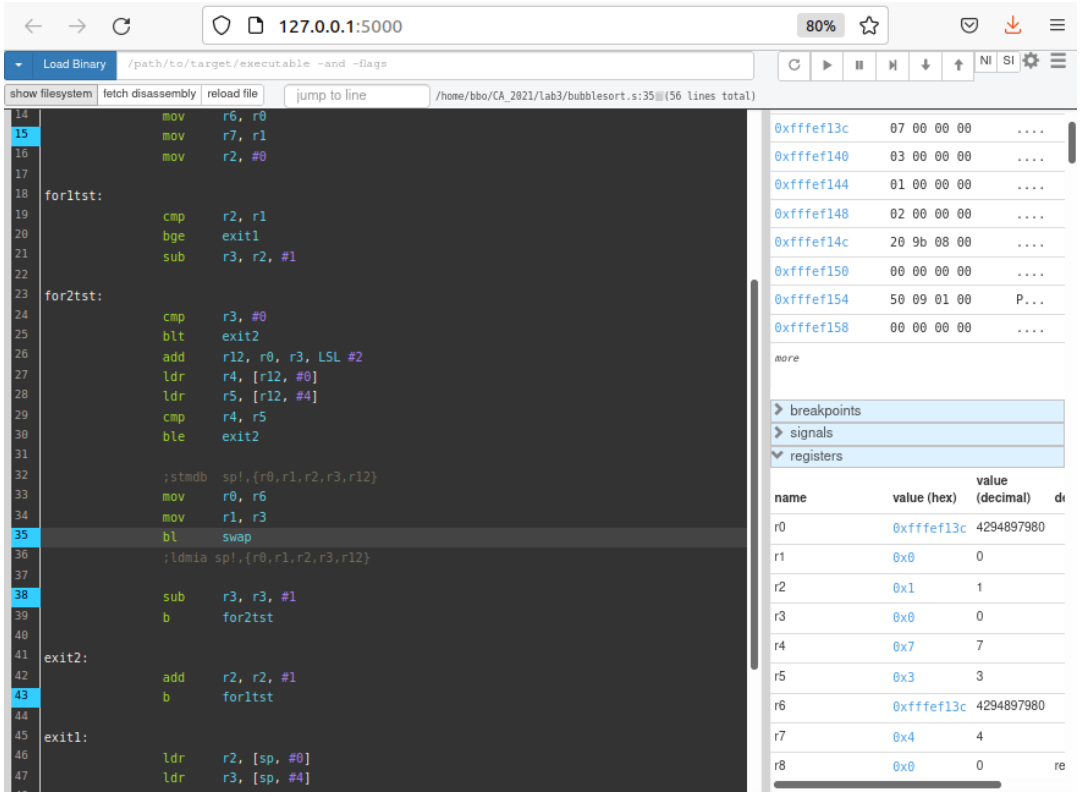
3) 실습 C



초기화면이다. arr에 7,3,1,2가 순서대로 들어있다.

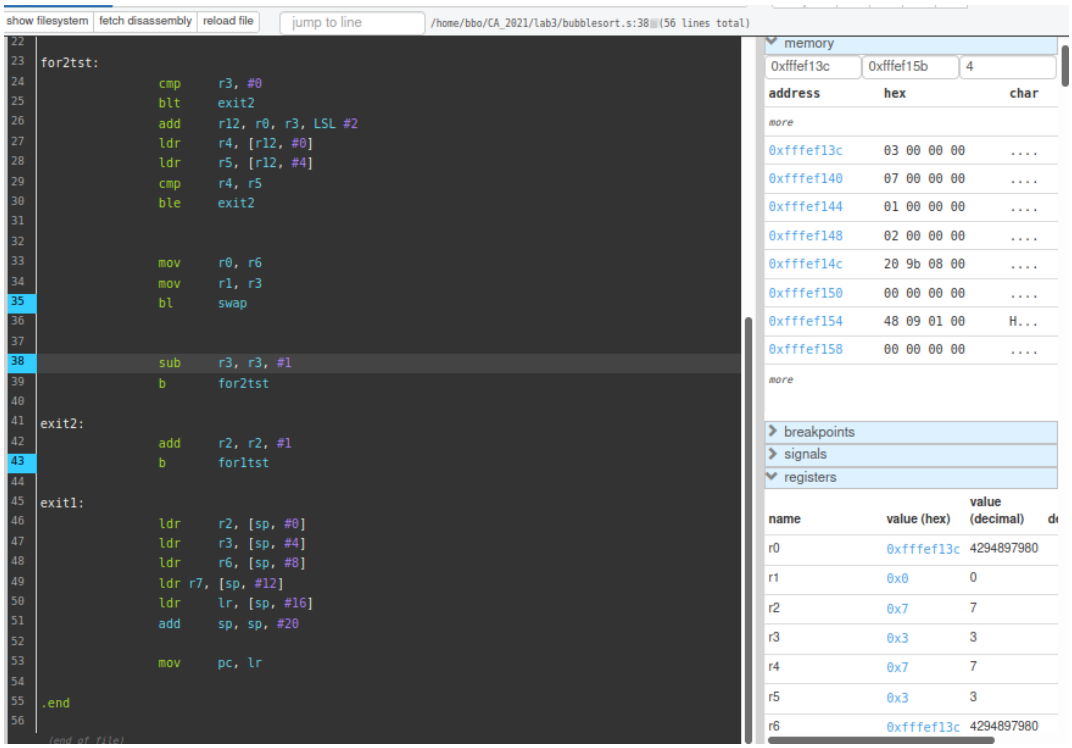


r2의 초기값은 0이었다. 따라서 outer loop에서 r3가 -1로 설정되고 inner loop에서 blt exit2로 branch되었다. add r2,r2,#1 명령어로 현재 r2는 1이 된 상태이다.



swap함수에 들어가기 전 상태이다.

r1과 r0는 각각 배열의 인덱스를 swap함수에서 사용하기위해 이전에 r0는 배열 시작주소, r1은 swap할 인덱스로 초기화 된 것을 볼 수 있다.



swap함수를 거친 상태다. 배열의 첫번째 방이었던 7과 배열의 두번째 방이었던 3이 서로 swap된

모습을 볼 수 있다.

r2, r3의 경우 각각 배열의 첫번째 데이터 7, 두번째 데이터 3이 된 것을 확인할 수 있다.

r0-r3,r12는 caller save로 호출하는 함수에서 호출 전 레지스터 상태를 저장해야 한다. 이 값들은 callee에서 원래 상태로 복구를 보장해주지 않기 때문이다. 이에 따라 swap함수에서는 r2, r3를 배열에 들어있는 데이터 값으로 활용하였고, bubblesort함수에서는 이전 상태를 저장해두고 복구하지 않아 swap함수에서 사용된 값이 그대로 넘어온 것을 확인할 수 있다.

4) 실습 D

변경된 코드

```
.text
.global bubblesort
.global swap
.type bubblesort, STT_FUNC

bubblesort:
                sub     sp, sp, #24
                str     lr, [sp, #20]
                str     r8, [sp, #16]
                str     r7, [sp, #12]
                str     r6, [sp, #8]
                str r3, [sp, #4]
                str     r2, [sp, #0]

                mov     r6, r0
                mov     r7, r1
                mov     r2, #0
                sub     r1, r1, #1

for1tst:
                cmp     r2, r1
                bge     exit1
                mov     r3, #0
                sub     r8, r1, r2

for2tst:
                cmp     r3, r8
                bge     exit2
                add     r12, r0, r3, LSL #2
                ldr     r4, [r12, #0]
                ldr     r5, [r12, #4]
                cmp     r4, r5
                ble     exit2
```

```

                stmdb   sp!,{r0,r1,r2,r3,r12}
                mov     r0, r6
                mov     r1, r3
                bl      swap
                ldmia sp!,{r0,r1,r2,r3,r12}

                add     r3, r3, #1
                b       for2tst

exit2:
                add     r2, r2, #1
                b       for1tst

exit1:
                ldr     r2, [sp, #0]
                ldr     r3, [sp, #4]
                ldr     r6, [sp, #8]
                ldr     r7, [sp, #12]
                str     r8, [sp, #16]
                ldr     lr, [sp, #20]
                add     sp, sp, #24

                mov     pc, lr

.end
(end of file)
```

결과

```
bbo@ubuntu:~/CA_2021/lab3$ qemu-arm -g 8080 ./lab3
Array before sorting : 7  3  1  2
Array after sorting : 1  2  3  7
bbo@ubuntu:~/CA_2021/lab3$
```